# DIGITAL SYSTEM DESIGN LABORATORY

## LAB 1

## IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG

## I. LAB OBJECTIVES

This Lab experiments are intended to implement Basic Combination Logic and Sequential Circuit in Verilog. Students are require to write test bench to simulate the given example code and Top level module to implement these codes in DE2-115 FPGA Kit.

a) For each experiment write the Verilog Code in three method ( dataflow, behavior and gate level)

b) For each model, write the Verilog Code to implement the these modules in DE2-FPGA Kit

(Show implementation results in Lab report)

c) For each model, Write the testbench to simulate this module.

(Do this task at home as a Home work 2 and show simulation results in Lab report)

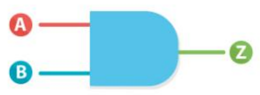
d) Analyze the FPGA implementation results and simulation results for these model

## II. PROCEDURE

### II.1 LAB EXPERIMENT 1 : WRITE HDL CODE TO REALIZE ALL LOGIC GATES

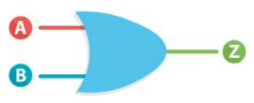
Refer to the tutorial "Quartus II Introduction Using Verilog Design" to create the first project in Quartus II. implement a simple circuit in DE2-115 Kit.

AND GATE



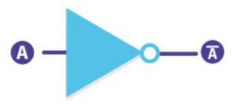
| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR GATE



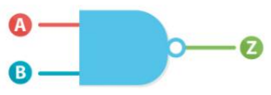
| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT GATE



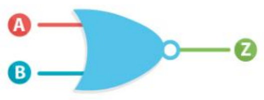
| A | $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

NAND GATE



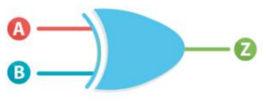
| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOR GATE



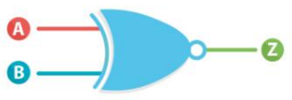
| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

XOR GATE



| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XNOR GATE



| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| Top module |
|---|

```
module Lab1_EX1(SW,LEDG,LEDR);
        input[17:0] SW;
        output[7:0] LEDG;
        output[17:0] LEDR;
        assign LEDR=SW;
        allgate_DF
DUT(SW[1],SW[0],LEDG[6],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);
endmodule
```

**Structural**

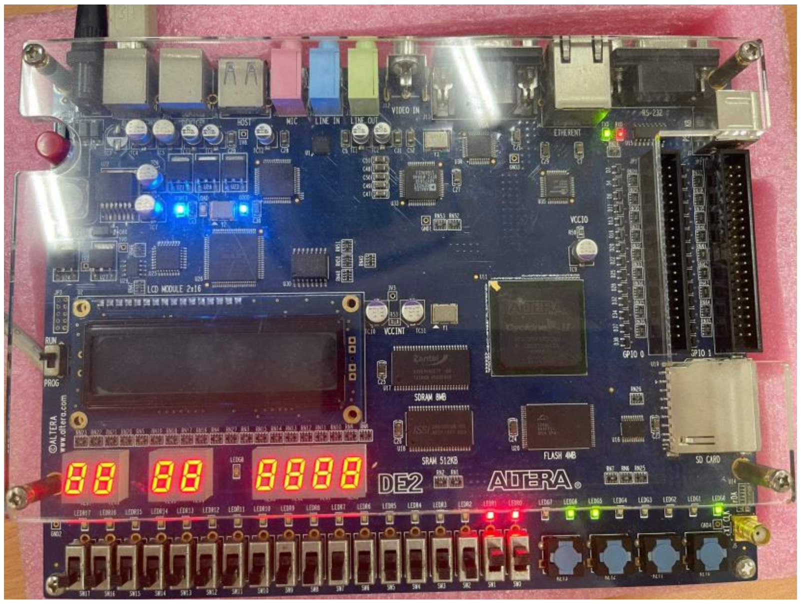
```
module allgate_GL ( a, b,
yand,yor,ynot,ynand,ynor,yxor,yx
nor );
        input a,b;
        output yand, yor, ynot,
ynand, ynor, yxor, yxnor;

        and   G1(yand,a,b);
        or    G2(yor,a, b);
        not   G3(ynot,a) ;
        nand G4 (ynand,a,b);
        nor   G5(ynor,a,b);
        xor   G6(yxor,a,b);
        xnor  G7(yxnor,a,b);

endmodule
```

**Dataflow**

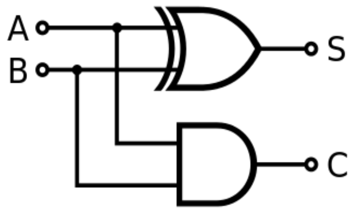
```
module allgate_DF ( a, b,
yand,yor,ynot,ynand,ynor,yxor,yx
nor );
        input a,b;
        output yand, yor, ynot,
ynand, ynor, yxor, yxnor;

        assign yand = a & b;
        assign yor = a | b;
        assign ynot = ~a ;
        assign ynand = ~(a & b);
        assign ynor = ~(a | b);
        assign yxor = a ^ b;
        assign yxnor =~(a^b);

endmodule
```

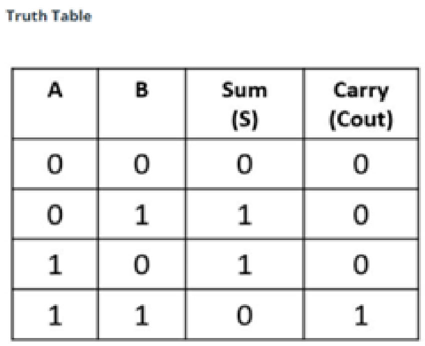
**Behavior**

```
module allgate_BH ( a, b,
yand,yor,ynot,ynand,ynor,yxor,yx
nor );
        input a,b;
        output reg  yand, yor,
ynot, ynand, ynor, yxor, yxnor
        always @(*) begin
            yand <= a & b;
            yor <= a | b;
            ynot <= ~a ;
            ynand <= ~(a & b);
            ynor <= ~(a | b);
            yxor <= a ^ b;
            yxnor <=~(a^b);
        end
endmodule
```



## II.2 LAB EXPERIMENT 2 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE HALF ADDER CIRCUIT:





Truth Table

| A | B | Sum (S) | Carry (Cout) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**Top module**

```
module Lab1_EX2(SW, LEDG, LEDR);
    input  [17:0] SW;
    output [7:0]  LEDG;
    output [17:0] LEDR;
    assign LEDR = SW;
    half_adder_DF DUT (
        .a(SW[1]),
        .b(SW[0]),
        .S(LEDG[0]),
        .Cout(LEDG[1])
    );
    assign LEDG[7:2] = 6'b0;
endmodule
```

**Structural**

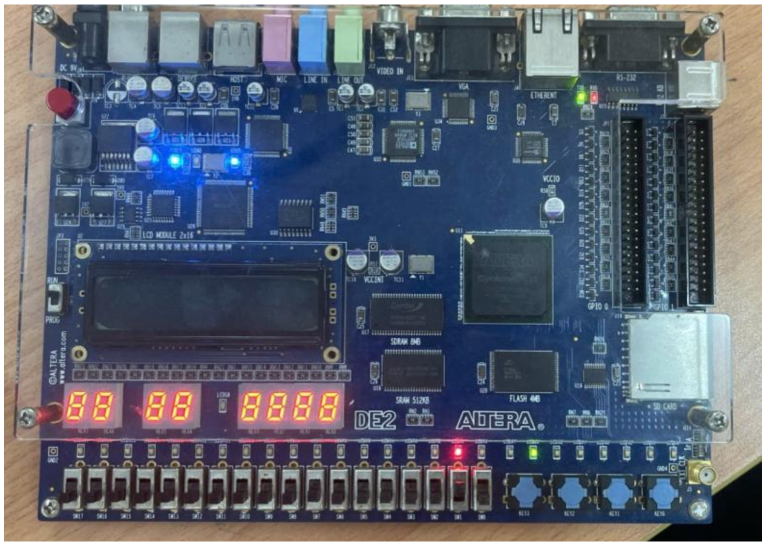
```
module half_adder_GL( a, b,  S,
Cout);
  input a,b;
  output  S,Cout;

  xor G1(S,a,b);
  and G2(Cout,a,b);
endmodule
```

**Dataflow**

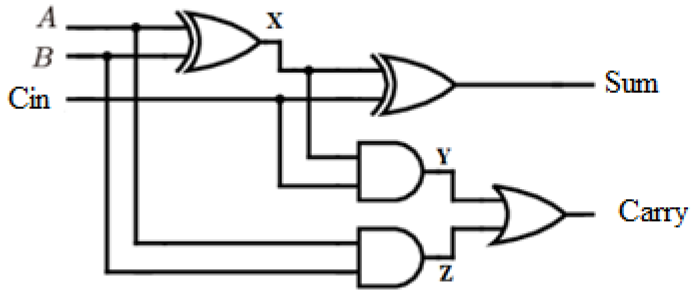
```
module half_adder_DF( a, b,  S,
Cout);
  input a,b;
  output S,Cout;

  assign S = a ^ b;
  assign Cout = a & b;
endmodule
```

**Behavior**

```
module half_adder_BH( a, b,  S,
Cout);
            input a,b;
            output reg S,Cout;

            always @(a,b) begin
                        S   <= a ^ b;
                        Cout <= a&b  ;
            end
endmodule
```



## II.3 EXPERIMENT 3: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL ADDER CIRCUIT:



| Inputs | | | Outputs | |
|---|---|---|---|---|
| $A$ | $B$ | $C_{in}$ | $S$ | $C_{out}$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Top module**

```
module Lab1_EX3(SW, LEDG, LEDR);
    input [17:0] SW;
    output [7:0] LEDG;
    output [17:0] LEDR;

    assign LEDR = SW;

    full_adder_GL FA1(
        .a(SW[0]),
        .b(SW[1]),
        .cin(SW[2]),
        .S(LEDG[0]),
        .Cout(LEDG[1])
    );
endmodule
```

**Structural**

```
module
full_adder_GL(a,b,cin,S,Cout);
            output S,Cout;
            input a,b,cin;
            wire x,y,z;

            xor g1(x,a,b);
    xor g2(S,x,cin);
            and g3(y,x,cin);
            and g4(z,a,b);
            or  g5(Cout,z,y);
endmodule
```

**Dataflow**

```
module full_adder_DF(a, b,
cin,S, Cout);
  input a,b,cin;
  output S, Cout;

  assign S = a ^ b ^ cin;
  assign Cout = (a & b) | (b &
cin) | (a & cin);
endmodule
```

**Behavior using always**

```
module
full_adder_BH3(a,b,cin,S,Cout);
            output reg S,Cout;
            input  a,b,cin;

            always @ (a,b,cin)
              begin
                        S   <= a^ b^cin;
                        Cout <=(a&b) |
(b&cin) | (cin&a);
              end
endmodule
```

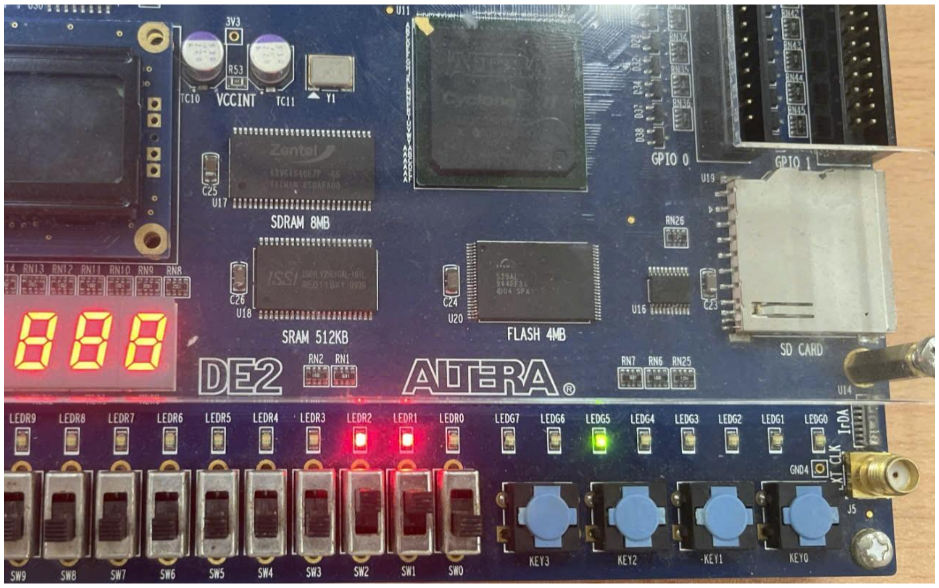
**Behavior using if else**

```
module full_adder_BH2( a, b, cin, S, Cout);
 input wire a, b, cin;
 output reg S, Cout;
 always @(a or b or cin) begin
            if(a==0 && b==0 && cin==0)
             begin
               S=0;Cout=0;
     end
 else if(a==0 && b==0 && cin==1)
  begin
   S=1;Cout=0;
  end
 else if(a==0 && b==1 && cin==0)
  begin
   S=1;Cout=0;
  end
 else if(a==0 && b==1 && cin==1)
  begin
   S=0;Cout=1;
  end
 else if(a==1 && b==0 && cin==0)
  begin
   S=1;Cout=0;
  end
 else if(a==1 && b==0 && cin==1)
  begin
   S=0;Cout=1;
  end
 else if(a==1 && b==1 && cin==0)
  begin
   S=0;Cout=1;
  end
 else if(a==1 && b==1 && cin==1)
```

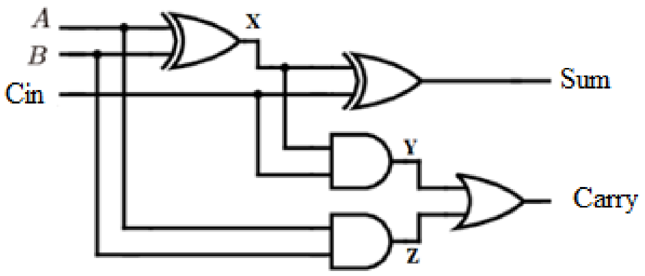
**Behavior using case**

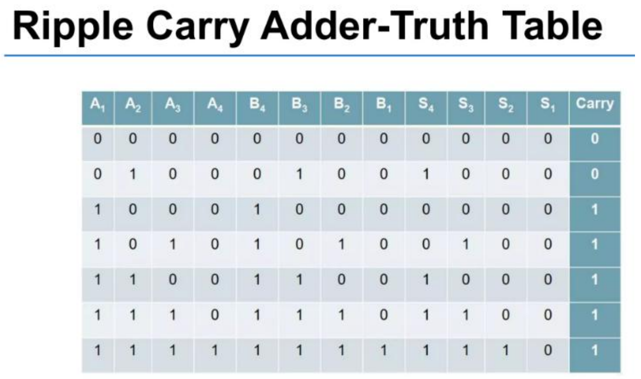
```
module full_adder_BH1 (a, b, cin,  S,  Cout);
 input a,b,cin;
 output reg S,Cout;
 always @(a or b or cin)
  begin

   case ({a , b , cin})
     3'b000: begin S = 0; Cout = 0; end
     3'b001: begin S = 1; Cout = 0; end
     3'b010: begin S = 1; Cout = 0; end
     3'b011: begin S = 0; Cout = 1; end
     3'b100: begin S = 1; Cout = 0; end
     3'b101: begin S = 0; Cout = 1; end
     3'b110: begin S = 0; Cout = 1; end
     3'b111: begin S = 1; Cout = 1; end
   endcase
  end
endmodule
```

```
  begin
   S=1;Cout=1;
  end
end
endmodule
```



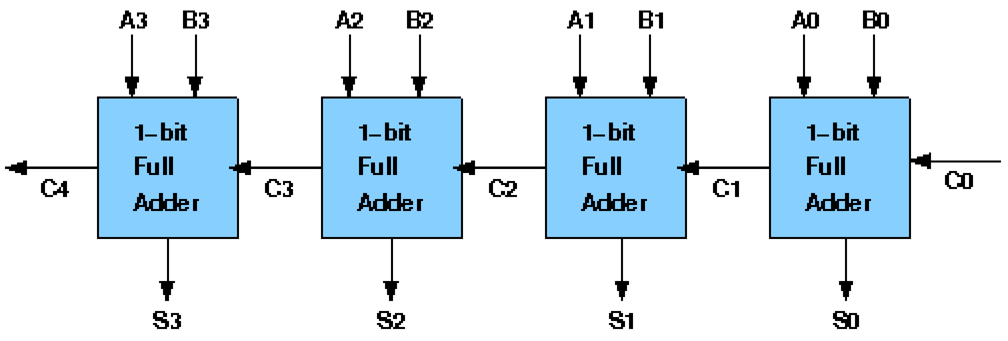
## II.4 EXPERIMENT 4 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE RIPPLE CARRY ADDER CIRCUIT:





### Ripple Carry Adder-Truth Table

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $S_4$ | $S_3$ | $S_2$ | $S_1$ | Carry |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |



**Top module**

```
module Lab1_ex4(SW,LEDG,LEDR);






endmodule
```

**Structural**

```
module
full_adder_STRU(a,b,cin,s,cout);
        output s,cout;
        input a,b,cin;
        wire x,y,z;
        xor g1(x,a,b);
        xor g2(s,x,cin);
        and g3(y,x,cin);
        and g4(z,a,b);
        or  g5(cout,y,z);
endmodule

module
four_bit_adder_STRU(cin,a,b,
s,cout);
        input [3:0] a,b;
        input cin;
        output [3:0] s;
        output cout;
        wire [2:0] w_carry;
        full_adder_STRU
C1(a[0],b[0],cin,s[0],
w_carry[0]);
        full_adder_STRU
C2(a[1],b[1], w_carry[0],s[1],
w_carry[1]);
        full_adder_STRU
C3(a[2],b[2], w_carry[1],s[2],
w_carry[2]);
        full_adder_STRU
C4(a[3],b[3], w_carry[2],s[3],
cout);
endmodule
```

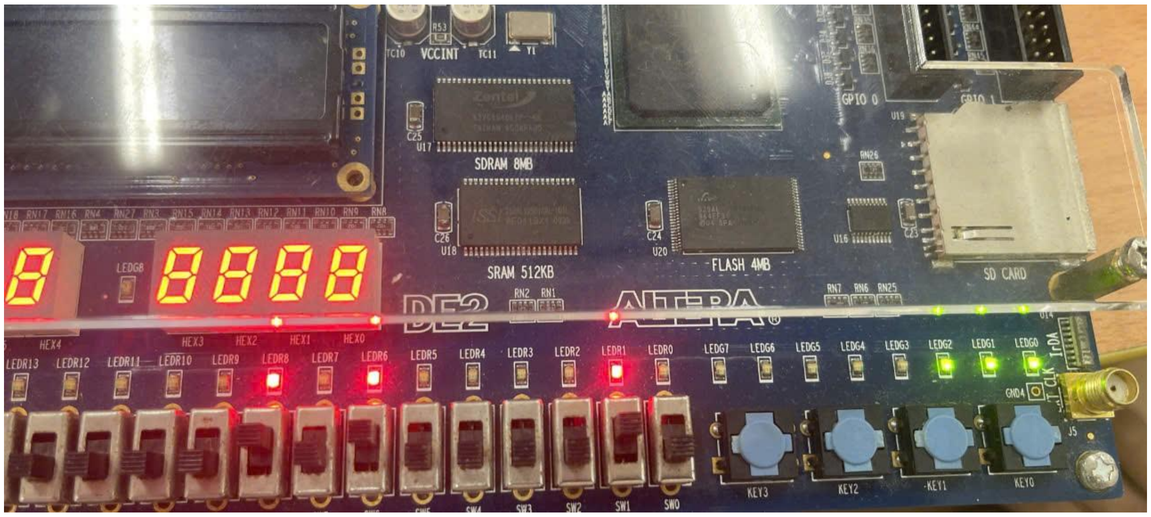
**Dataflow**

```
module
full_adder_DF(a,b,cin,s,cout);
   input a, b, cin;
   output Sum, Carry;

   assign Sum = a ^ b ^ cin;
   assign Carry = (a & b) | (b & cin)
| (cin & a);
endmodule

module
four_bit_adder_DF(a,b,cin,s,cout);
   input [3:0] a, b;
   input cin;
   output [3:0] s;
   output cout;

   wire [3:0] carry;

   // Assign carry for each bit
   assign carry[0] = (a[0] & b[0]) |
(b[0] & cin) | (cin & a[0]);
   assign carry[1] = (a[1] & b[1]) |
(b[1] & carry[0]) | (carry[0] &
a[1]);
   assign carry[2] = (a[2] & b[2]) |
(b[2] & carry[1]) | (carry[1] &
a[2]);
   assign carry[3] = (a[3] & b[3]) |
(b[3] & carry[2]) | (carry[2] &
a[3]);

   // Assign sum for each bit
   assign s[0] = a[0] ^ b[0] ^ cin;
   assign s[1] = a[1] ^ b[1] ^
carry[0];
   assign s[2] = a[2] ^ b[2] ^
carry[1];
```

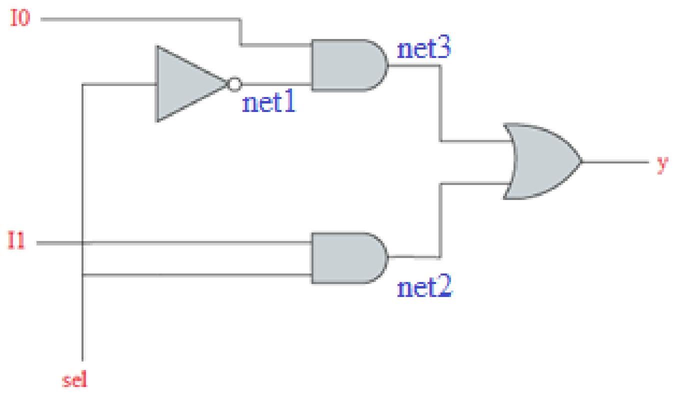
**Behavior**

```
module half_adder_BH( a, b,  s,
cout);
     input a,b;
     output reg s,cout;

     always @(a,b) begin
          s  <= a ^ b;
          cout <= a&b ;
     end
endmodule

module
four_bit_adder_BH(a,b,cin,s,co
ut);
   input [3:0] a, b;
   input cin;
   output reg [3:0] s;
   output reg cout;

   reg [3:0] carry;

   always @(*) begin
      // Compute sums and
carries for each bit using non-
blocking assignments
      s[0] <= a[0] ^ b[0] ^ cin;
      carry[0] <= (a[0] & b[0]) |
(b[0] & cin) | (cin & a[0]);

      s[1] <= a[1] ^ b[1] ^
carry[0];
      carry[1] <= (a[1] & b[1]) |
(b[1] & carry[0]) | (carry[0] &
a[1]);

      s[2] <= a[2] ^ b[2] ^
carry[1];
      carry[2] <= (a[2] & b[2]) |
```

| | | |
|---|---|---|
| | assign s[3] = a[3] ^ b[3] ^ carry[2];<br><br>// Assign final carry-out<br>assign cout = carry[3];<br>endmodulex | (b[2] & carry[1]) \| (carry[1] & a[2]);<br><br>s[3] <= a[3] ^ b[3] ^ carry[2];<br>cout <= (a[3] & b[3]) \| (b[3] & carry[2]) \| (carry[2] & a[3]);<br>end<br>endmodule |



## II.5 EXPERIMENT 5 :WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 2:1 MULTIPLEXER CIRCUIT:



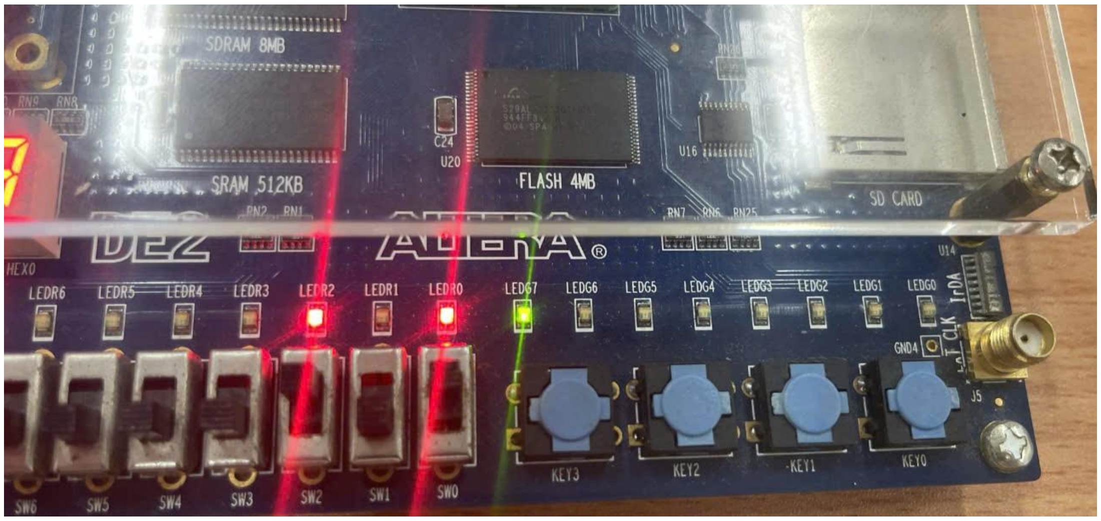
**Truth Table**

| sel | y |
|---|---|
| 0 | $i_0$ |
| 1 | $i_1$ |

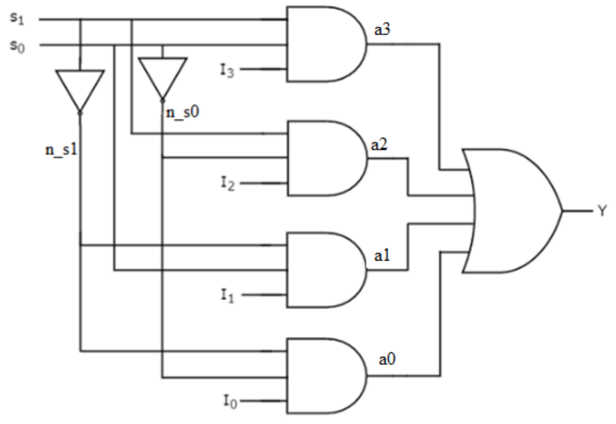
| **Top module** |
|---|
| module Lab1_EX5(SW,LEDG,LEDR);<br>input[17:0] SW;<br>output[7:0] LEDG;<br>output [17:0] LEDR; |

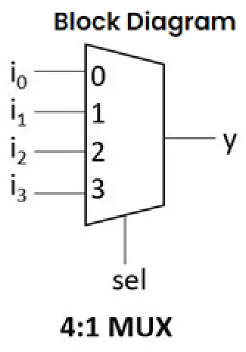
```
        assign LEDR=SW;

        //mux21_DF(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));
        //mux21_BEH(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));
        mux21_GL(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));
endmodule
```

| **Structural** | **Dataflow** | **Behavior** |
|---|---|---|
| module mux21_GL(i,sel,y);<br>    input sel;<br>    input [1:0] i;<br>    output y;<br>    wire net1,net2,net3;<br>    not g1(net1,sel);<br>    and g2(net2,i[1],sel);<br>    and g3(net3,i[0],net1);<br>    or g4(y,net3,net2);<br>endmodule | module mux21_DF(i,sel,y);<br>    input sel;<br>    input [1:0] i;<br>    output y;<br>    assign y<br>=(i[0]&(~sel))\|(i[1]&sel);<br>endmodule | module mux21_BEH(i,sel,y);<br>    input sel;<br>    input [1:0] i;<br>    output reg y;<br>    always@(*) begin<br>    if(sel==0) y=i[0];<br>    if(sel==1)y=i[1];<br>    end<br>endmodule |



## II.6 EXPERIMENT 6 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 4:1 MULTIPLEXER CIRCUIT:

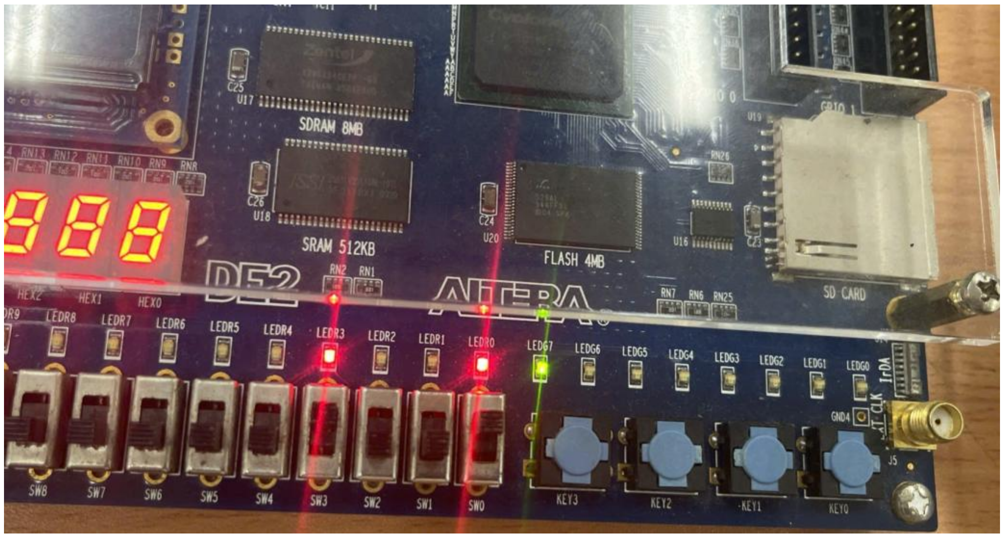




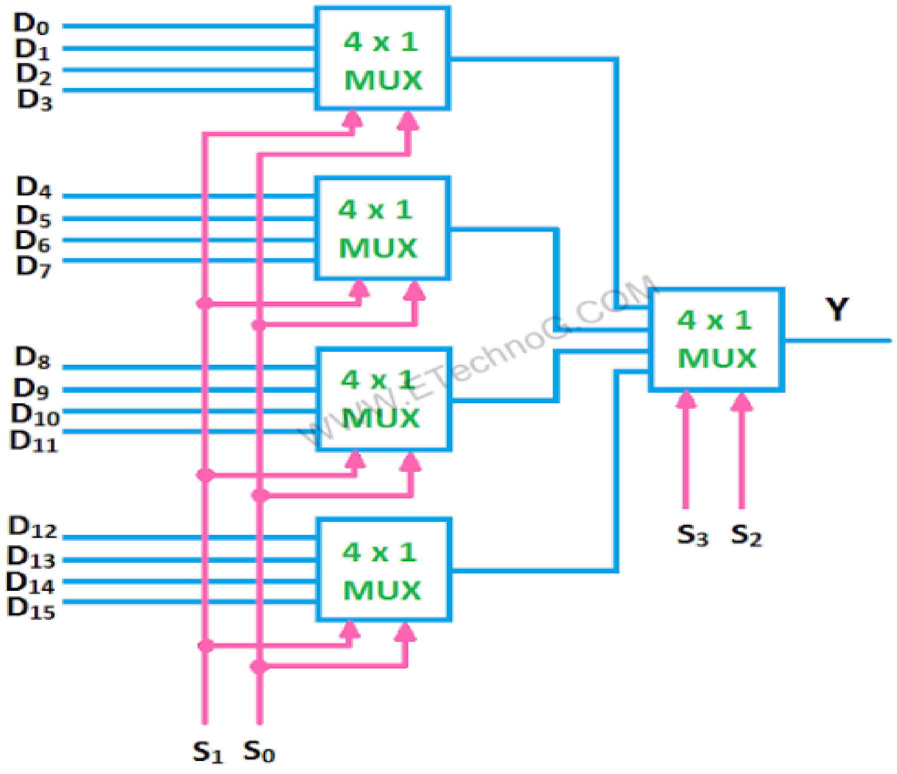
Truth Table

| sel[0] | sel[1] | y |
|---|---|---|
| 0 | 0 | $i_0$ |
| 0 | 1 | $i_1$ |
| 1 | 0 | $i_2$ |
| 1 | 1 | $i_3$ |

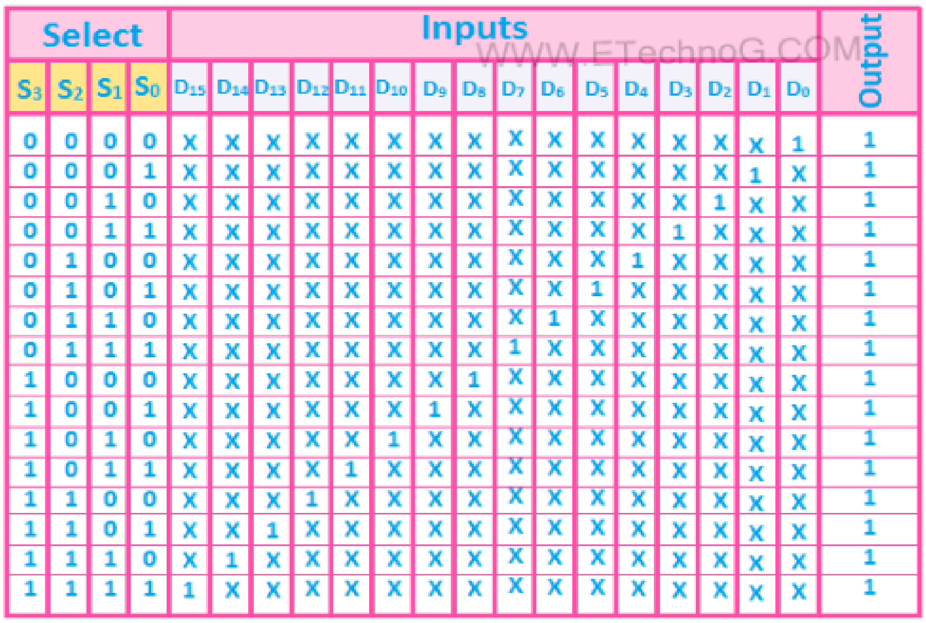
| | | |
|---|---|---|
| **Top module** | | |
| module Lab1_EX6(SW,LEDR,LEDG);<br><br><br><br><br>endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module mux41_GL(i,s,y);<br>  input [3:0] i;<br>  input [1:0] s;<br>  output y;<br>  wire [1:0] n_s;<br>  wire [3:0]a;<br>  not g0(n_s[0],s[0]);<br>  not g1(n_s[1],s[1]);<br>  and<br>g2(a[0],i[0],n_s[0],n_s[1]);<br>  and<br>g3(a[1],i[1],n_s[1],s[0]);<br>  and<br>g4(a[2],i[2],s[1],n_s[0]);<br>  and g5(a[3],i[3],s[1],s[0]);<br>  or  g6(y,a[0],a[1],a[2],a[3]);<br>endmodule | module mux41_df<br>(i0,i1,i2,i3,s0,s1,y);<br>        input<br>i0,i1,i2,i3,s0,s1;<br>        output y;<br>        assign y=<br>i0&(~s1)&(~s0) \| i1<br>&(~s1)&s0 \|<br>i2&s1&(~s0) \|<br>i3&s1&s0;<br>endmodule | module mux41_BEH_v1(i,s,y );<br>  output reg y ;<br>  input [3:0] i ;<br>  input [1:0] s ;<br>  always @ (i,s) begin<br>        if (s[1]==0&s[0]==0)<br>        y <= i[0];<br>        else if (s[1]==0&s[0]==1)<br>        y <= i[1];<br>        else if (s[1]==1&s[0]==0)<br>        y <= i[2];<br>        else<br>        y <= i[3];<br>   end<br>endmodule<br><br>**Version 2:**<br>module mux41_BEH_v2(i,s,y );<br>        output reg y ;<br>        input [3:0]i;<br>        input [1:0]s;<br> always@(i,s) begin<br>       case ({s[1],s[0]})<br>        2'b00: y <= i[0];<br>        2'b01: y <= i[1];<br>        2'b10: y <= i[2];<br>        2'b11: y <= i[3];<br>       endcase<br> end<br>endmodule |



## II.7 EXPERIMENT 7 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 16:1 MULTIPLEXER CIRCUIT USING STUCTURAL MODEL FROM FIVE 4:1 MULTIPLEXER MODULE



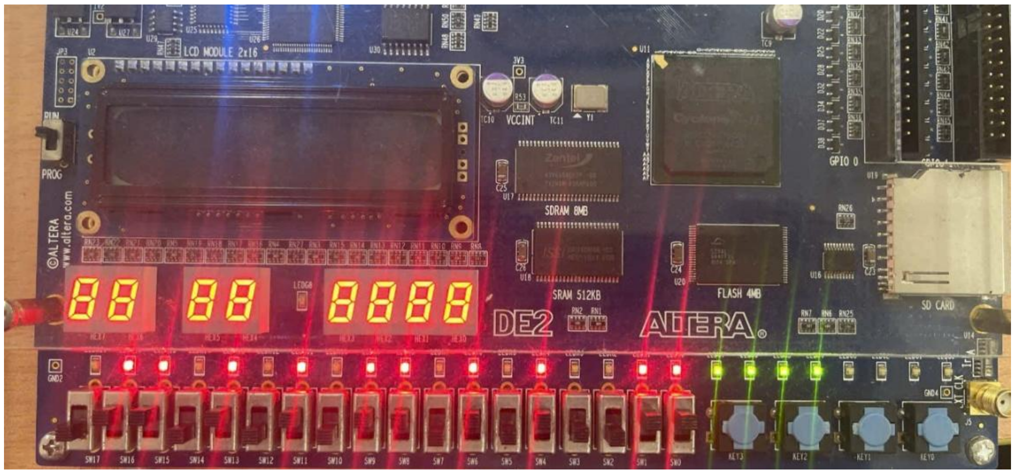
16 to 1 Multiplexer



| Select | | | | Inputs | | | | | | | | | | | | | | | | Output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | $D_{15}$ | $D_{14}$ | $D_{13}$ | $D_{12}$ | $D_{11}$ | $D_{10}$ | $D_9$ | $D_8$ | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | |
| 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 | 1 |
| 0 | 0 | 0 | 1 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 | X | 1 |
| 0 | 0 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 | X | X | 1 |
| 0 | 0 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | X | X | 1 | X | X | X | 1 |
| 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | 1 | X | X | X | X | 1 |
| 0 | 1 | 0 | 1 | X | X | X | X | X | X | X | X | X | X | 1 | X | X | X | X | X | 1 |
| 0 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | 1 | X | X | X | X | X | X | 1 |
| 0 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | 1 | X | X | X | X | X | X | X | 1 |
| 1 | 0 | 0 | 0 | X | X | X | X | X | X | X | 1 | X | X | X | X | X | X | X | X | 1 |
| 1 | 0 | 0 | 1 | X | X | X | X | X | X | 1 | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 0 | 1 | 0 | X | X | X | X | X | 1 | X | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 0 | 1 | 1 | X | X | X | X | 1 | X | X | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 1 | 0 | 0 | X | X | X | 1 | X | X | X | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 1 | 0 | 1 | X | X | 1 | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 1 | 1 | 0 | X | 1 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 |
| 1 | 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 1 |

**Answer code:**

```
module Lab1_EX7(SW,LEDG,LEDR,KEY);
        input [17:0] SW;
        input [3:0] KEY;
        output [7:0] LEDG;
        output [17:0] LEDR;

        assign LEDR = SW;
        assign LEDG[7:4] = KEY;

        mux16_1(.i0(KEY[3:0]),.i(SW[11:0]),.s(SW[17:14]),.y(LEDG[0]));
endmodule

module mux41_GL(i,s,y);
        input [3:0] i;
        input [1:0] s;
        output y;
        wire n_s0,n_s1, a0,a1,a2,a3;
        not g0(n_s0,s[0]);
        not g1(n_s1,s[1]);
        and g2(a0,i[0],n_s0,n_s1);
        and g3(a1,i[1],n_s1,s[0]);
        and g4(a2,i[2],s[1],n_s0);
        and g5(a3,i[3],s[1],s[0]);
        or  g6(y,a0,a1,a2,a3);
endmodule

module mux16_1(i0,i,s,y);
        input [3:0] i0;
        input [11:0] i;
        input [3:0] s;
        output y;
        wire [3:0]a;

        mux41_GL(i0[3:0],s[1:0],a[0]);
        mux41_GL(i[3:0],s[1:0],a[1]);
        mux41_GL(i[7:4],s[1:0],a[2]);
        mux41_GL(i[11:8],s[1:0],a[3]);

        mux41_GL(a[3:0],s[3:2],y)
endmodule
```



## IV. LAB REPORT GUIDELINES

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Circuit Schematics, Truth Table, Verilog Module Codes, Verilog test bench codes, Top level module to implement the required circuit in FPGA KIT and evidences of data output evidences to validate the experiments  (The Captured Screens, Photo of FPGA Kit implementation results).